```python
import numpy as np

'''
Simple (sample) Python code to estimate power generation using Jensen model
for a Wind Farm:
  Handles multiple turbines, Detects wake overlap, Applies quadratic wake
  superposition, Uses a customizable power curve, Turbines are assumed
  aligned in wind direction.

  Assumption: Wake expands linearly downstream the turbine defined by wake
    decay constant 'k'

  User inputs require: Free-stream wind velocity U_inf, Turbines as an array
  of list - defined by class Turbine.
  (x_loc, y_loc, rotor_diameter, Thrust coefficient and power curve formula),
  cut-in, cut-out and rated speed of the wind turbines.

  Features: Computes wake deficits, Combines multiple wakes, Calculates
  effective wind speed at each turbine, Computes power output by interpolating
  velocity between rated and cut-in conditions.

  Further extension planned: Option to use Power Coefficient curve, Annual
  Energy Production (AEP) Calculation, Calculation of Capacity Factor (CF)
  where CF = (Actual AEP) / (Rated Power × 8,760 hours) × 100%
'''
# ----------------------------
# Turbine and Model Parameters
# ----------------------------
v_cut_in, v_rated, v_cut_out = 3.0, 11.4, 25.0 # [m/s]
rated_power = 2.00  # [MW]

class Turbine:
  def __init__(self, x, y, rotor_diameter, Ct, power_curve):
    self.x = x
    self.y = y
    self.D = rotor_diameter
    self.R = rotor_diameter / 2
    self.Ct = Ct
    self.power_curve = power_curve  # function(U) -> Power in MW

# ----------------------------
# Jensen Wake Model Functions
# ----------------------------
def wake_radius(R, k, dx):
  return R + k * dx   # R * (1 + 2.k.dx/D)

def velocity_deficit(U_inf, Ct, R, k, dx):
  '''
    Compute velocity deficit using Jensen model, k is wake decay constant,
    Ct is thurst coefficient.
  '''
  wake_rad = wake_radius(R, k, dx)
  return U_inf * (1 - np.sqrt(1 - Ct)) / (wake_rad / R) ** 2

def is_in_wake(turbine_i, turbine_j, k):
  '''
    Check if turbine_j is in wake of turbine_i
  '''

  dx = turbine_j.x - turbine_i.x
  if dx <= 0:
    return False

  dy = abs(turbine_j.y - turbine_i.y)
  wake_r = wake_radius(turbine_i.R, k, dx)

  return dy <= wake_r

# ----------------------------
# Farm Wake Calculation
# ----------------------------
def compute_effective_wind_speed(turbines, U_inf, k):

  n = len(turbines)
  U_eff = np.full(n, U_inf)

  for j in range(n):
    deficits_sq = []

    for i in range(n):
      if i == j:
        continue

      if is_in_wake(turbines[i], turbines[j], k):
        dx = turbines[j].x - turbines[i].x

        dU = velocity_deficit(U_inf, turbines[i].Ct, turbines[i].R, k, dx)
        deficits_sq.append((dU / U_inf) ** 2)

    if deficits_sq:
      total_deficit = np.sqrt(sum(deficits_sq))
      U_eff[j] = U_inf * (1 - total_deficit)

  return U_eff

def air_density_humid(pressure, temperature, relative_humidity):
    '''
    Calculate air density including humidity effects
    '''
    R_d = 287.05     # Gas constant for dry air
    R_v = 461.495    # Gas constant water vapour

    # Saturation vapour pressure (Tetens formula)
    T_C = temperature - 273.15
    e_s = 610.94 * np.exp((17.625 * T_C) / (T_C + 243.04))

    # Actual vapour pressure
    e = relative_humidity * e_s

    # Partial pressures
    p_d = pressure - e

    rho = (p_d / (R_d * temperature)) + (e / (R_v * temperature))
    return rho

def power_turbine(U, R, rho=1.187, Cp=0.3):
    A = np.pi * R**2
    return 0.5 * rho * A * Cp * U**3

# ----------------------------
# Power Calculation
# ----------------------------
def compute_power_output(turbines, U_eff):
  power = []
  for turb, U in zip(turbines, U_eff):
    power.append(turb.power_curve(U))

  return np.array(power)


def wt_power_curve(U):
  '''
    Calculate power generated by Wind Turbine in [MW]
  '''
  if U < v_cut_in or U > v_cut_out:
    return 0.0
  elif U < v_rated:
    return rated_power * ((U - v_cut_in) / (v_rated - v_cut_in)) ** 3
  else:
    return rated_power

# ------------------------------------------------------
# Example calculation of Power Generation in a Wind Farm
# ------------------------------------------------------
if __name__ == "__main__":
  # Free-stream wind speed in [m/s]
  U_inf = 11.4

  # Wake decay constant: offshore ~0.04, onshore ~0.075
  k = 0.04

  # Define turbines (aligned in wind direction)
  turbines = [
    Turbine(0,     0, 126, 0.8, wt_power_curve),
    Turbine(2000,  0, 126, 0.8, wt_power_curve),
    Turbine(3000,  0, 126, 0.8, wt_power_curve),
  ]

  # Compute effective wind speeds
  U_eff = compute_effective_wind_speed(turbines, U_inf, k)

  # Compute power
  power = compute_power_output(turbines, U_eff)

  # Results
  for i, (u, p) in enumerate(zip(U_eff, power)):
    print(f"Turbine {i+1}: Effective Free-stream Velocity = {u:5.2f} m/s, \
Power = {p:7.3f} MW")

  print(f"\nTotal Farm Power: {np.sum(power):7.3f} MW")
```